

Programación 1



Tema 4

Programación Modular




Departamento de Ciencia de la
Computación e Inteligencia Artificial

Objetivos / Competencias

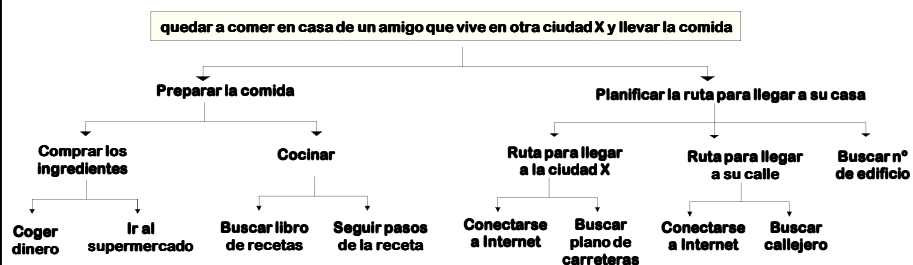
1. Utilizar el diseño descendente para resolver problemas de relativa complejidad
2. Comprender las diferencias entre procedimientos y funciones
3. Saber modularizar programas en lenguaje C

Índice

1. Descomposición modular 
2. Comunicación entre módulos
3. Procedimientos y Funciones
4. Ámbito de las variables
5. Estructura general de un programa
6. Funciones predefinidas en lenguaje C
7. Fuentes de información

Diseño descendente (“top-down”)

- ♦ Para resolver un problema se **divide** en problemas más pequeños (**subproblemas**)
 - La descomposición del problema se realiza en una serie de niveles o pasos sucesivos de **refinamiento**, que forman una **estructura jerárquica**
 - Cada nivel de la jerarquía incluye un mayor nivel de detalle



Concepto de Módulo

- ◆ Cuando un programa es grande y complejo no es conveniente que todo el código esté dentro del programa principal (función main() en lenguaje C)
- ◆ Un módulo o **subprograma** ...
 - es un bloque de código que se escribe aparte del programa principal
 - se encarga de realizar una tarea concreta que resuelve un problema parcial del problema principal
 - puede ser invocado (llamado) desde el programa principal o desde otros módulos
 - permite ocultar los detalles de la solución de un problema parcial (**Caja negra**)

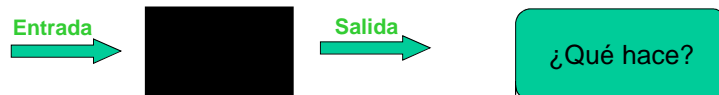
Programación 1. Dto. CCIA. Curso 2013-14

P-5

Caja negra

- ◆ Cada módulo es una caja negra para el programa principal o para el resto de módulos
- ◆ Para utilizar un módulo desde el programa principal o desde otros módulos ...

- Necesitamos conocer su **interfaz**, es decir, sus entradas y salidas



- No necesitamos conocer los **detalles internos de funcionamiento**



Programación 1. Dto. CCIA. Curso 2013-14

P-6

Ejemplo de módulos

```
main() {
    int n1, n2; // números introducidos por teclado (datos de entrada)
    int mayor; // el mayor número de los 2 introducidos (dato de salida)
    int menor; // el menor número de los 2 introducidos (dato de salida)

    cout << "Introduce dos números enteros: ";
    cin >> n1 >> n2;
    mayor = maximo(n1, n2);
    menor = minimo(n1, n2);
    cout << "El mayor número es:" << mayor;
    cout << "El menor número es:" << menor;
    cout << endl;
}
```



¿Qué hace el módulo
"maximo()"?

```
// Este módulo devuelve el menor de dos números
int minimo(int a, int b)
{
    int m; // el menor de dos números (dato de salida)

    m = a;
    if (b < m)
        m = b;
    return(m);
}
```

```
// Este módulo devuelve el mayor de dos números
int maximo(int a, int b)
{
    int m; // el mayor de dos números (dato de salida)

    if (a > b)
        m = a;
    else
        m = b;
    return(m);
}
```



¿Cómo lo
hace?

Programación 1. Dto. CCIA. Curso 2013-14

P-7

Declaración, definición y llamada de un módulo

Declaración del módulo

Nombre_del_módulo (declaración_de_parámetros)

int maximo(int a, int b);

Definición del módulo

Nombre_del_módulo (declaración_de_parámetros)

Declaración_de_variables_locales

Cuerpo del módulo: sentencias ejecutables

Fin_del_módulo

```
int maximo(int a, int b)
{
    int m;

    if (a > b)
        m = a;
    else
        m = b;
    return(m);
}
```

Llamada del módulo

Nombre_del_módulo (lista_de_parámetros)

mayor = maximo(n1, n2);


Programación 1. Dto. CCIA. Curso 2013-14

P-8

Ventajas de la programación modular

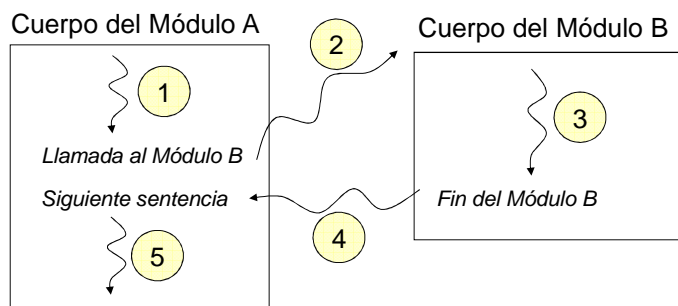
- 💧 Facilita el diseño descendente y la programación estructurada
- 💧 Reduce el tiempo de programación
 - ❑ *Reusabilidad* : estructuración en *librerías* específicas (biblioteca de módulos)
 - ❑ División de la tarea de programación entre un equipo de programadores
- 💧 Disminuye el tamaño total del programa
 - ❑ Un módulo sólo está escrito una vez y puede ser utilizado varias veces desde distintas partes del programa
- 💧 Facilita la detección y corrección de errores
 - ❑ mediante la comprobación individual de los módulos
- 💧 Facilita el mantenimiento del programa
 - ❑ Los programas son más fáciles de modificar
 - ❑ Los programas son más fáciles de entender (más legibles)

Índice

1. Descomposición modular
- 2. Comunicación entre módulos** 
3. Procedimientos y Funciones
4. Ámbito de las variables
5. Estructura general de un programa
6. Funciones predefinidas en lenguaje C
7. Fuentes de información

Transferencia del flujo de control

- Cuando un módulo A llama (invoca) a otro módulo B, el flujo de control (flujo de ejecución) pasa al módulo B
- Cuando termina de ejecutarse el módulo B, el flujo de control continúa en el módulo A, a partir de la sentencia siguiente a la llamada del módulo B



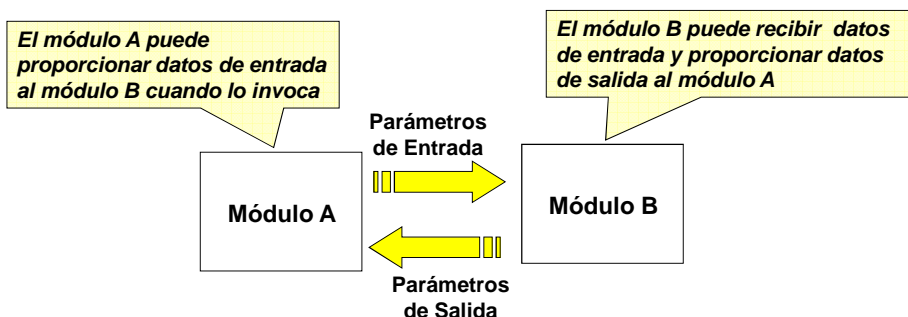
El programa principal puede ser considerado como un módulo que puede invocar a otros módulos, pero que no puede ser invocado por ningún módulo

Programación 1. Dto. CCIA. Curso 2013-14

P-11

Transferencia de información

- La transferencia de información entre módulos se realiza a través del **paso de parámetros o argumentos**
- Un módulo puede tener parámetros de entrada y/o de salida



Programación 1. Dto. CCIA. Curso 2013-14

P-12

Parámetros actuales y formales

Parámetros actuales o reales

- Los que aparecen en la sentencia de llamada al módulo

Nombre_del_módulo (pr1, pr2, ..., prN)

mayor = maximo(**n1**, **n2**);



Parámetros formales o ficticios

- Los que aparecen en la declaración del módulo

Nombre_del_módulo (tipo1 pf1, tipo2 pf2, ..., tipoN pfN)

int maximo(int **a**, int **b**);



Correspondencia entre parámetros actuales y formales:

- ✓ número de parámetros
- ✓ tipo de parámetros
- ✓ orden de los parámetros
- ✗ ~~nombre de los parámetros~~

Programación 1. Dto. CCIA. Curso 2013-14

P-13

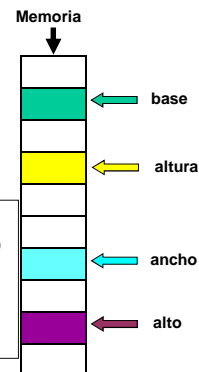
Paso de parámetros por Valor

- El módulo recibe una copia del valor del dato (parámetro actual) que el módulo invocador le pasa
- El parámetro actual puede ser cualquier expresión evaluable en el momento de la llamada al módulo
- Si dentro del módulo se modifica el parámetro formal correspondiente, el valor del parámetro actual permanece inalterable

```
main() {  
    int base, altura, area, perimetro;  
  
    cout << "Díme la base del rectángulo:";  
    cin >> base;  
    cout << "Díme su altura:";  
    cin >> altura;  
  
    rectangulo(base, altura, area, perimetro);  
  
    cout << "Area: " << area << endl;  
    cout << "Perímetro: " << perimetro;  
    cout << endl;  
}
```

Paso de parámetros por Valor

```
void rectangulo(int ancho, int alto,  
               int &area_rect, int &perim )  
{  
    area_rect = ancho * alto;  
    perim = 2 * (ancho + alto);  
}
```



Programación 1. Dto. CCIA. Curso 2013-14

P-14

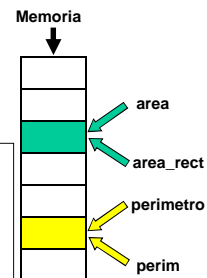
Paso de parámetros por Referencia

- El módulo recibe la referencia a la posición de memoria donde se encuentra dicho valor (dirección de memoria de una variable)
- El parámetro actual debe ser obligatoriamente una variable (que puede contener o no un valor)
- Si dentro del módulo se modifica el parámetro formal correspondiente, se estará cambiando el contenido en memoria del parámetro actual

```
main() {  
    int base, altura, area, perimetro;  
  
    cout << "Díme la base del rectángulo:";  
    cin >> base;  
    cout << "Díme su altura:";  
    cin >> altura;  
  
    rectangulo(base, altura, area, perimetro);  
  
    cout << "Área: " << area << endl;  
    cout << "Perímetro: " << perimetro;  
    cout << endl;  
}
```

Paso de parámetros por Referencia

```
void rectangulo( int ancho, int alto,  
                int &area_rect, int &perim )  
{  
    area_rect = ancho * alto;  
    perim = 2 * (ancho + alto);  
}
```



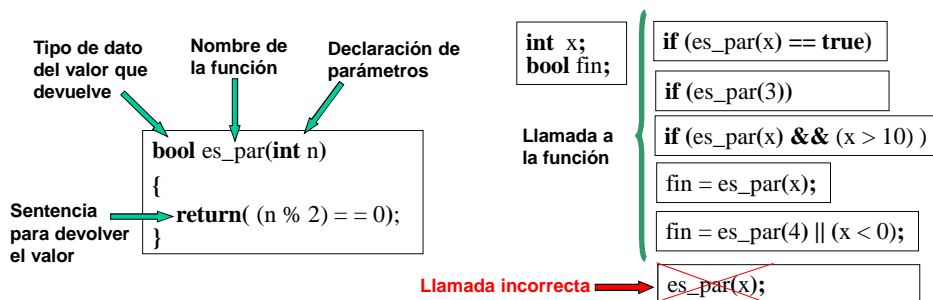
Índice

1. Descomposición modular
2. Comunicación entre módulos
3. Procedimientos y Funciones
4. Ámbito de las variables
5. Estructura general de un programa
6. Funciones predefinidas en lenguaje C
7. Fuentes de información



Concepto de Función

- Devuelve un valor que está asociado al nombre de la función
- Se suele definir con N parámetros ($N \geq 1$)
- Sólo se debe utilizar paso de **parámetros por valor**

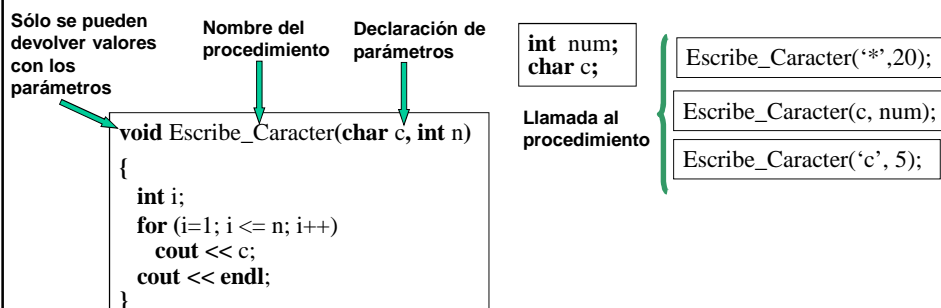


Programación 1. Dto. CCIA. Curso 2013-14

P-17

Concepto de Procedimiento

- Se puede definir con N parámetros ($N \geq 0$)
- Se puede utilizar paso de **parámetros por valor y/o por referencia**
- Se invoca con una sentencia compuesta por su nombre y lista de parámetros actuales (la llamada es una sentencia por sí misma)



Programación 1. Dto. CCIA. Curso 2013-14

P-18

¿Qué uso: un procedimiento o una función?

¿El módulo tiene que **devolver un solo valor**?

Si

función

No


procedimiento

Sobre la sentencia return

return (*expresión*);

- ◆ Finaliza la ejecución del cuerpo de la función
- ◆ Se encarga de devolver el valor de retorno de la función, después de evaluar su *expresión* asociada
- ◆ Es recomendable usar una sola sentencia return dentro del cuerpo de una función
- ◆ Debería ser la última sentencia del cuerpo de la función

Índice

1. Descomposición modular
2. Comunicación entre módulos
3. Procedimientos y Funciones
- 4. Ámbito de las variables** 
5. Estructura general de un programa
6. Funciones predefinidas en lenguaje C
7. Fuentes de información

Concepto de ámbito de una variable

- El ámbito de una variable define la visibilidad de la misma, es decir, desde dónde se puede acceder a dicha variable

```
main() {  
    int n; // número introducido por teclado (dato de entrada)  
  
    cout << "Introduce un número entero: ";  
    cin >> n;  
    if (es_primo(n))  
        cout << "El número es primo";  
    else  
        cout << "El número no es primo";  
    cout << endl;  
}
```

ámbito de **n**

```
// Este módulo comprueba si un número es primo o no  
bool es_primo(int num)  
{  
    int cont; // contador (dato auxiliar)  
    bool primo; // es primo o no (dato de salida)  
  
    primo = true;  
    cont = 2;  
    while ( (cont < num) && primo) {  
        // comprobar si es divisible por otro número  
        primo = ! (num % cont == 0);  
        cont = cont + 1;  
    }  
    return (primo);  
}
```

ámbito de
num, cont, primo

Variables locales y variables globales

Variable local

- ❑ Su ámbito es el cuerpo del módulo en donde está declarada
- ❑ Se crea cuando se declara y se destruye cuando finaliza la ejecución del módulo

Variable global

- ❑ Su ámbito es todo el programa (todos sus módulos y el programa principal)
- ❑ Se crea cuando se declara y se destruye cuando finaliza la ejecución del programa

Prohibido utilizar variables globales

La **comunicación** entre módulos debe realizarse a través de parámetros, y **NO de variables globales**



Efecto lateral

- Cualquier comunicación de datos entre módulos al margen de los parámetros y la devolución de resultados se denomina efecto lateral

```
#include <iostream>
using namespace std;

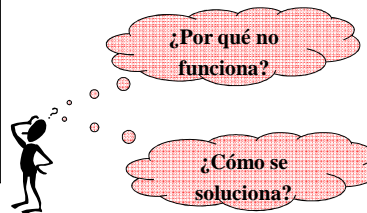
int resultado; // declaración de variable global

int Num_Mayor(int n1, int n2);

main() {
    int n1, n2; // números introducidos por teclado (datos de entrada)
    int mayor; // el mayor de los dos números (dato de salida)

    cout << "Introduce dos números :";
    cin >> n1 >> n2;
    resultado = n1 + n2;
    mayor = Num_Mayor(n1, n2);
    cout << "La suma de los dos números es: " << resultado;
    cout << " y el mayor de ellos es:" << mayor;
    cout << endl;
}
```

```
// función que devuelve el mayor de dos números
int Num_Mayor(int n1, int n2)
{
    if (n1 > n2)
        resultado = n1;
    else
        resultado = n2;
    return(resultado);
}
```



Índice

1. Descomposición modular
2. Comunicación entre módulos
3. Procedimientos y Funciones
4. Ámbito de las variables
- 5. Estructura general de un programa**
6. Funciones predefinidas en lenguaje C
7. Fuentes de información



¿Qué tipo de programas debo ser capaz de hacer?

#directivas del preprocesador

Declaración de constantes

Declaración de procedimientos y funciones

main() {

Declaración de variables (de tipos simples)

Cuerpo principal

sentencias de control

llamadas a procedimientos y funciones

}

Definición de procedimientos y funciones

Programación 1. Dto. CCIA. Curso 2013-14

P-27

Ejemplo de programa

```
#include <iostream>
using namespace std;

// Cambios de moneda a Euros
const float US_DOLAR_EURO = 1,4696;
const float LIBRA_ESTERLINA_EURO = 1,4696;

// Declaración de procedimientos y funciones
void Leer_Importe(float &cantidad, char &moneda);
float Cambio_En_Euros(float cantidad, char moneda);

main() {
    float cantidad; // cantidad de dinero (dato de entrada)
    char moneda; // tipo de moneda (dato de entrada)
    char respuesta; // respuesta para continuar (dato de entrada)
    float euros; // cantidad en euros equivalente (dato de salida)

    do {
        Leer_Importe(cantidad, moneda);
        euros = Cambio_En_Euros(cantidad, moneda);
        cout << "El cambio en euros es:" << euros << endl;
        cout << "¿Desea introducir otro importe? (S/N) :";
        cin >> respuesta;
    } while ( (respuesta == 's') || (respuesta == 'S') );
}

// Leer de teclado un importe y el tipo de moneda
// validando los datos introducidos hasta que
// sean correctos
void Leer_Importe(float &cantidad, char &moneda)
{
    bool datos_correctos;
    do {
        cout << "Introduce cantidad de dinero y moneda (D/L):";
        cin << cantidad << moneda;
        datos_correctos = (cantidad > 0.0) &&
            (moneda == 'D' || moneda == 'L');
    } while ( ! datos_correctos );
}

// Devolver el cambio en euros equivalente al importe y moneda
// especificados
float Cambio_En_Euros(float cantidad, char moneda)
{
    switch (moneda) {
        case 'D': euros = cantidad * US_DOLAR_EURO;
            break;
        case 'L': euros = cantidad * LIBRA_ESTERLINA_EURO;
    }
    return (euros);
}
```



Cuando defines un módulo, recuerda incluir un **comentario** que explique **qué hace** el módulo

Programación 1. Dto. CCIA. Curso 2013-14

P-28

Índice

1. Descomposición modular
2. Comunicación entre módulos
3. Procedimientos y Funciones
4. Ámbito de las variables
5. Estructura general de un programa
- 6. Funciones predefinidas en lenguaje C**
7. Fuentes de información



Bibliotecas del lenguaje C / C++

- ♦ La mayoría de lenguajes de programación proporcionan una colección de procedimientos y funciones de uso común (**bibliotecas** o **librerías**)
- ♦ En lenguaje C / C++, para hacer uso de los módulos incluidos en una biblioteca se utiliza la directiva del compilador **#include**
- ♦ Existe una gran variedad de bibliotecas disponibles:
 - ☐ Funciones matemáticas
 - ☐ Manejo de caracteres y de cadenas de caracteres
 - ☐ Manejo de entrada y salida de datos
 - ☐ Manejo del tiempo (fecha, hora, ...)
 - ☐ etc.

Algunas funciones predefinidas en lenguaje C / C++

Librería C++	Librería C	Función	Descripción
<math.h>	<math.h>	double cos(double x)	Devuelve el coseno de x
		double sin(double x)	Devuelve el seno de x
		double exp(double x)	Devuelve e ^x
		double fabs(double x)	Devuelve el valor absoluto de x
		double pow(double x, double y)	Devuelve x ^y
		double round(double x)	Devuelve el valor de x redondeado
		double sqrt(double x)	Devuelve la raíz cuadrada de x
<iostream>	<ctype.h>	int isalnum(int c)	Devuelve verdadero si el parámetro es una letra o un dígito
		int isdigit(int c)	Devuelve verdadero si el parámetro es un dígito
		int toupper(int c)	Devuelve el carácter en mayúsculas
	<stdlib.h>	int rand(void)	Devuelve un número aleatorio entre 0 y RAND_MAX

Librería C++	Librería C	Constantes	Descripción
<iostream>	<stdint.h>	INT_MIN	Menor número entero representable
		INT_MAX	Mayor número entero representable

Programación 1. Dto. CCIA. Curso 2013-14

P-31

Ejercicios

1. Cuando hace frío, los meteorólogos informan sobre un índice llamado *factor de enfriamiento por el viento* que tiene en cuenta la velocidad de viento y la temperatura. Este factor se puede aproximar con la fórmula:

$$W = 13.12 + 0.6215 \cdot t - 11.37 \cdot v^{0.16} + 0.3965 \cdot t \cdot v^{0.016}$$

donde v = velocidad del viento en m/s
 t = temperatura en grados Celsius: $t \leq 10$
 W = índice de enfriamiento del viento (en grados Celsius)

Diseña un módulo que solicite el valor de la velocidad y la temperatura, y calcule W , teniendo en cuenta la restricción impuesta para la temperatura.
2. Diseña un módulo que reciba como parámetro un número n y dibuje en pantalla un cuadrado de tamaño n formado por asteriscos.
3. Mejora el ejercicio 2 añadiendo otro parámetro que permita que el cuadrado se dibuje con el carácter enviado como parámetro.
4. ¿Se te ocurre alguna forma para indicar que el cuadrado se dibuje hueco o sólido?
5. Diseña un módulo que permita leer y validar un dato de entrada de manera que su valor sea mayor que 0 y menor que 100 y devuelva la suma y la cuenta de los números entre 1 y dicho valor.

Programación 1. Dto. CCIA. Curso 2013-14

P-32

Índice

1. Descomposición modular
2. Comunicación entre módulos
3. Procedimientos y Funciones
4. Ámbito de las variables
5. Estructura general de un programa
6. Funciones predefinidas en lenguaje C

7. Fuentes de información



Bibliografía Recomendada


Fundamentos de Programación
Jesús Carretero, Félix García, y otros
Thomson-Paraninfo 2007. ISBN: 978-84-9732-550-9

 Capítulo 7

Problemas Resueltos de Programación en Lenguaje C
Félix García, Alejandro Calderón, y otros
Thomson (2002) ISBN: 84-9732-102-2

 Capítulo 5

Resolución de Problemas con C++
Walter Savitch
Pearson Addison Wesley 2007. ISBN: 978-970-26-0806-6

 Capítulo 4